Robot Programming #11

Communication

Dept. of Mech. Robotics and Energy Eng.
Dongguk University

Serial Communication

- Serial is a generic protocol used by computers and electronic modules to send and receive control information and data.
- The Serial link has two unidirectional channels, one for sending and one for receiving.
- The link is asynchronous, and so both ends of the serial link must be configured to use the same settings.
- One of the Serial connections goes via the USB port, allowing you to easily communicate with your host PC.

Serial Communication

- The Arduino sketch has a built-in serial monitor that can communicate with the Arduino through a USB port over the same USB cable that is used for programming.
- This enables you to:
  - Print out messages to a host PC terminal (useful for debugging!)
  - Read input from the host PC keyboard
  - Communicate with applications and programming languages running on the host PC that can communicate with a serial port, e.g. perl, python, java and so on.

Serial monitor example

- Connect a LED and a resistor as shown below.

Serial monitor example

- Write the following program and upload it.

```
void setup() {
  pinMode(11, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter 1 to LED On or any to LED Off");
}

void loop() {
   if (Serial.available()) {
      char ch = Serial.read();
      if (ch=="1") {
         digitalWrite(rLED,1);
         Serial.println("LED On");
      }
      else {
         digitalWrite(rLED,0);
         Serial.println("LED Off");
      }
   }
}
```

A Closer Look

- Serial.available() determines whether there is a data in the serial link.

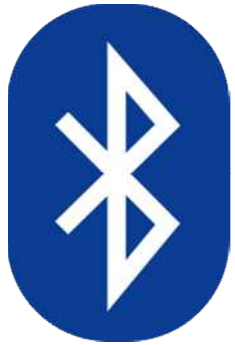
```
Serial.available()
```

- Serial.read() reads an input from the serial link.

```
char ch = Serial.read();
      if (ch==“1”) {
```

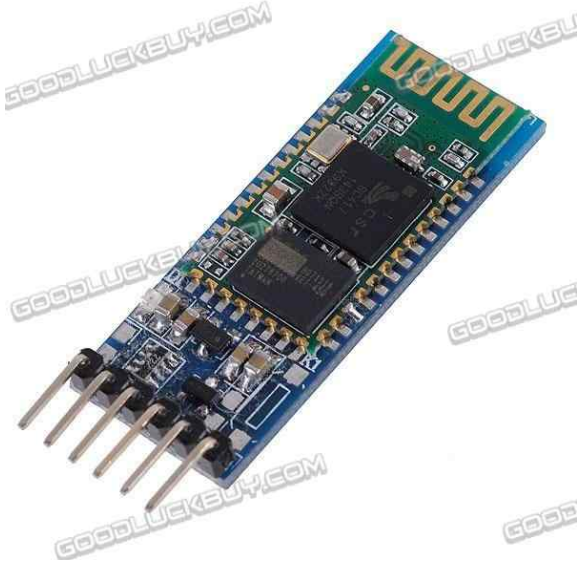
Bluetooth Communication

- a wireless technology standard for exchanging data over short distances (using short-wavelength radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices.
- originally conceived as a wireless alternative to RS-232 data cables.
- It can connect several devices, overcoming problems of synchronization.



Bluetooth Slave UART Board

- EPX4P4YA (from eleparts.co.kr)

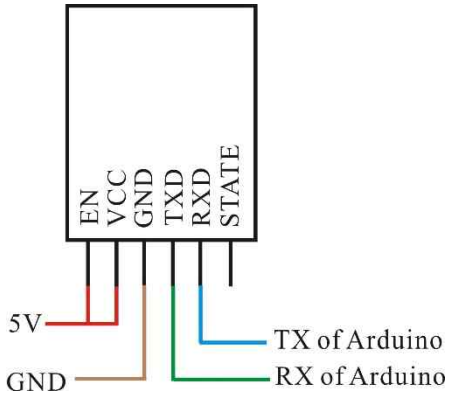


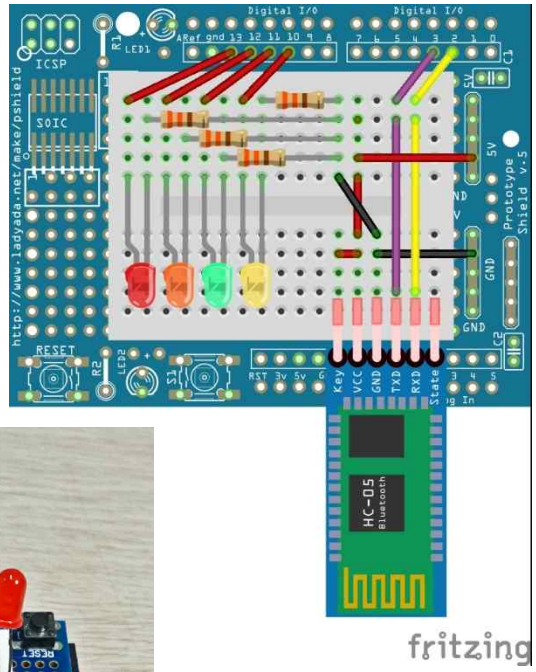
Bluetooth module

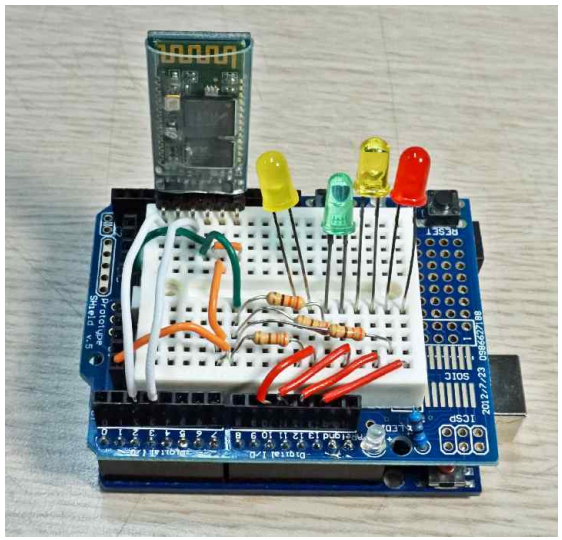
- The Bluetooth module has 6 legs.
- TXD: Transmit Data (Data output pin for the UART),must be connect to the RXD of your device.
- RXD: Receive Data (Data input pin for the UART),must be connect to the TXD of your device.
- VCC: power supply, between 3.6V~6.0V,Prohibit more than 7V,Otherwise, damage to device!
- GND: Ground
- State: Connection status output, when bluetooth is connect, output high, otherwise, output pulse(about 5Hz)
- EN: enable the module

Bluetooth Connection

- Connect the FRDM-KL25Z to the Arduino and 4 LEDs to pins 10-13..

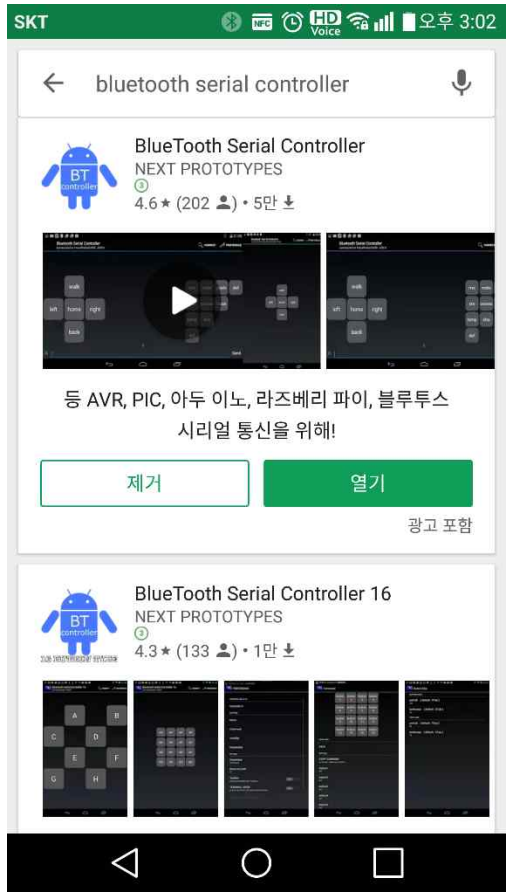






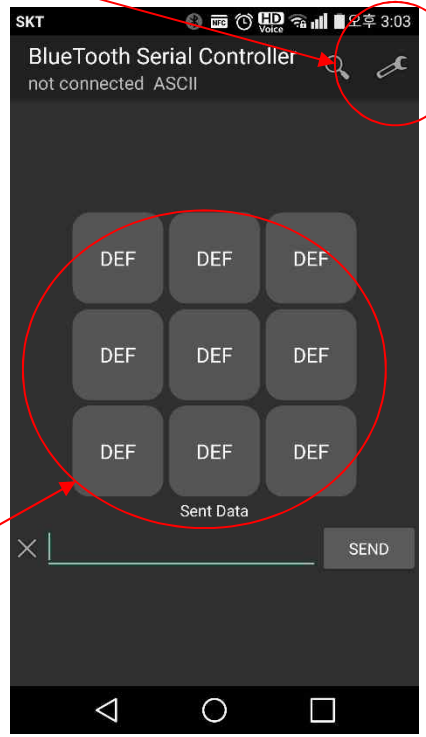
Bluetooth Serial Comm App

- The status LED of the Bluetooth module will flicker.
- Let's control the Arduino by a cellular phone.
- Install App(BlueTooth Serial Controller).
- Turn on the Bluetooth and connect to HC-06(PW:1234).
- After connection, the status LED will be on.



Bluetooth Connection

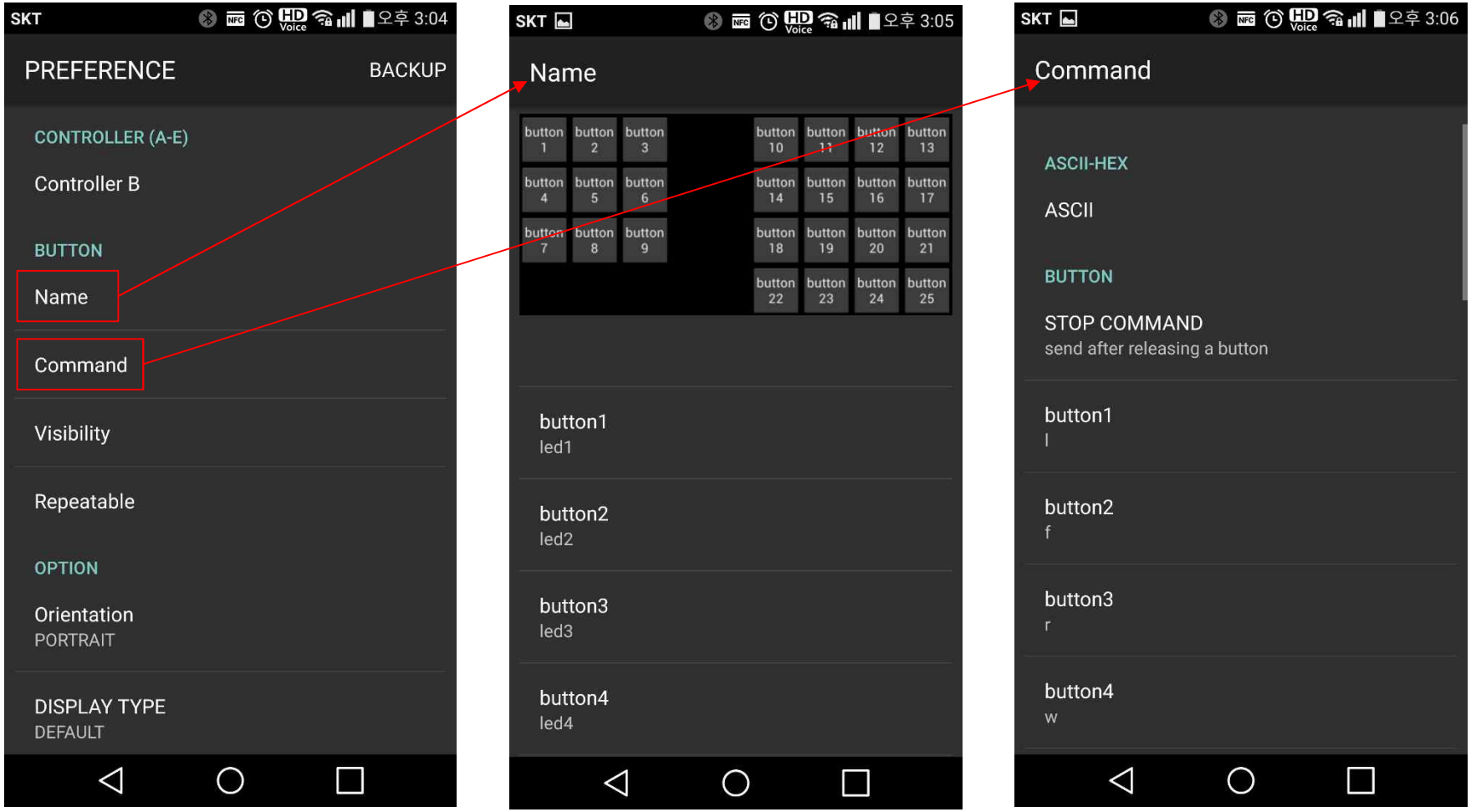
- We are going to use control mode.
- Click setup button and configure settings.



We are going to use these buttons first.

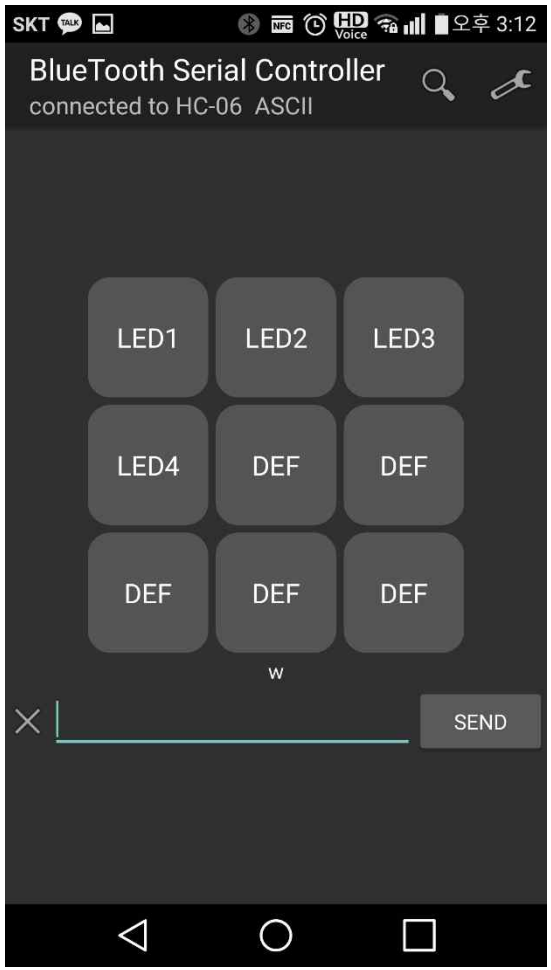
Bluetooth Connection

- Configure buttons like below.



Bluetooth Connection

- Configure buttons like below.



Bluetooth Test program

- Write a code as shown and upload it.

```
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(3 , 2);

int led1 = 13;
int led2 = 12;
int led3 = 11;
int led4 = 10;
int a = 0;
int b = 0;
int c = 0;
int d = 0;
void setup() {
  BTSerial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}
```

```
void loop() {
    byte data;
    data = BTSerial.read();
    if(data=='l'){
      a = !a;
      digitalWrite(led1, a);
    }
    if(data=='f'){
      b = !b;
      digitalWrite(led2, b);
    }
    if(data=='r'){
      c = !c;
      digitalWrite(led3, c);
    }
    if(data=='w'){
      d = !d;
      digitalWrite(led4, d);
    }
}
```

Bluetooth Test

- If you connect the Bluetooth UART board, the status LED of the board will flash. This means that the board is waiting for connection.
- Turn on the Bluetooth of the smart phone and wait for connection.
- If connected, the status LED of the board will stays on.
- Run the Bluetooth Serial Comm App and your Arduino is now ready for action.
- Give it a try!

A Closer Look

- SoftwareSerial.h library is included. BTSerial is configured to have TX and RX pin numbers.

```
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(3 , 2);
```

- BTSerial.begin() initializes the serial link.

```
BTSerial.begin(9600);
```

- BTSerial.read reads the serial link data.

```
byte data;
data = BTSerial.read();
```

- ! is used to invert the logical data.

```
a = !a;
```

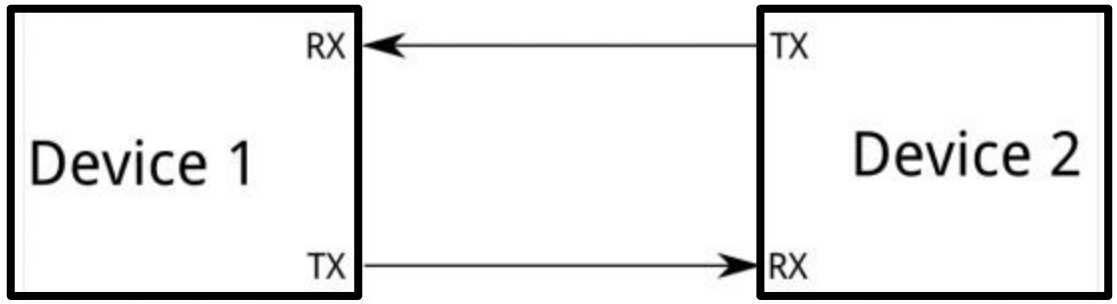
I2C

- I2C (Inter-Integrated Circuit) is pronounced *I-two-C.* Alternatively I2C is spelled **I²C** (pronounced I-squared-C) or **IIC** (pronounced I-I-C).
- I2C is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to communicate with one or more “master” chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.

Main Features of I2C

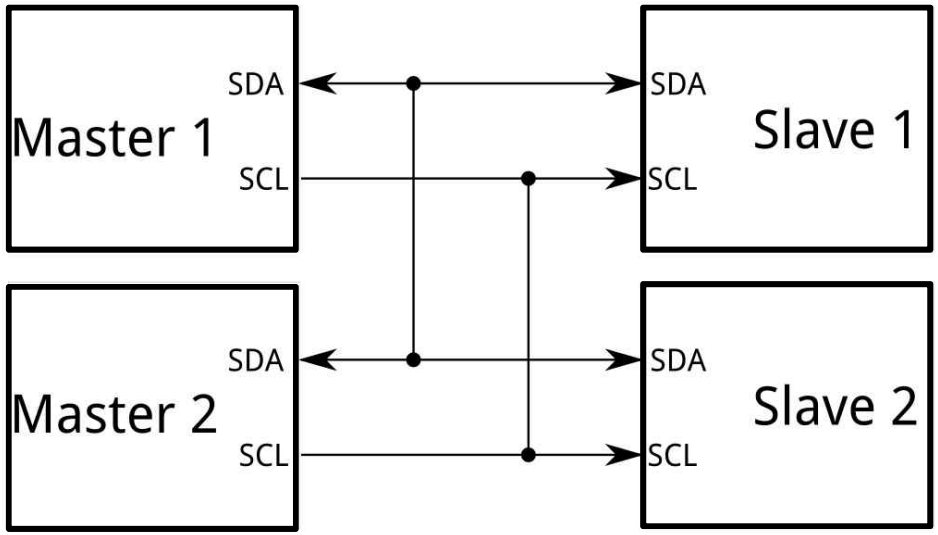
- Most significant features include:
  - Only two bus lines are required
  - No strict baud rate requirements like for instance with RS232, the master generates a bus clock
  - Simple master/slave relationships exist between all components.
  - Each device connected to the bus is software-addressable by a unique address
  - I2C is a true multi-master bus providing arbitration and collision detection

What's wrong with Serial Ports?



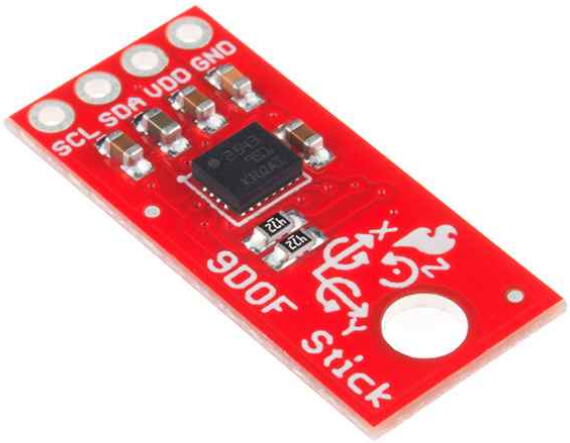
- Because serial ports are **asynchronous** (no clock data is transmitted), devices using them must agree ahead of time on a data rate.
- The two devices must also have clocks that are close to the same rate, and will remain so.
- they are inherently suited to communications between two, and only two devices.

Why use I2C?



- I^2C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. Also, unlike SPI, I^2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus

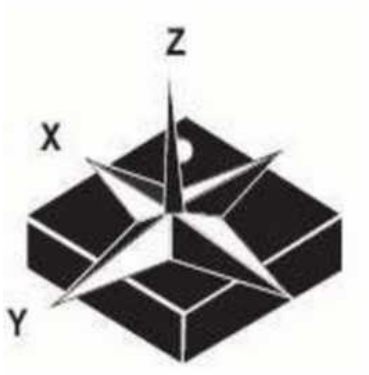
9DoF IMU Module

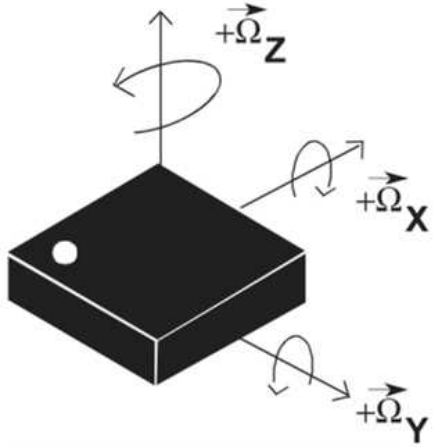


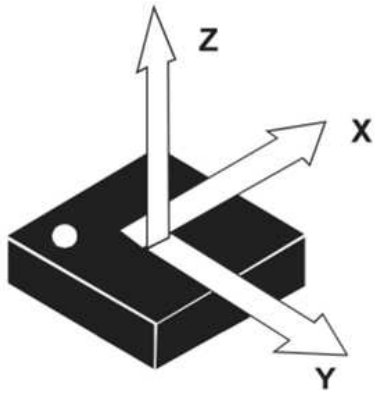
LSM9DS1 9 Degree of Freedom IMU Module

Features:

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
- ±2/±4/±8/±16g linear acceleration full scale
- ±4/±8/±12/±16 gauss magnetic full scale
- ±245/±500/±2000dps angular rate full scale
- I2C serial interface
- Operating Voltage: 3.3V

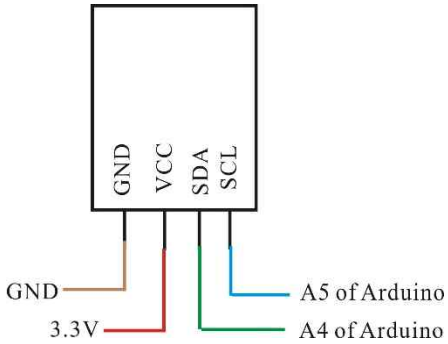




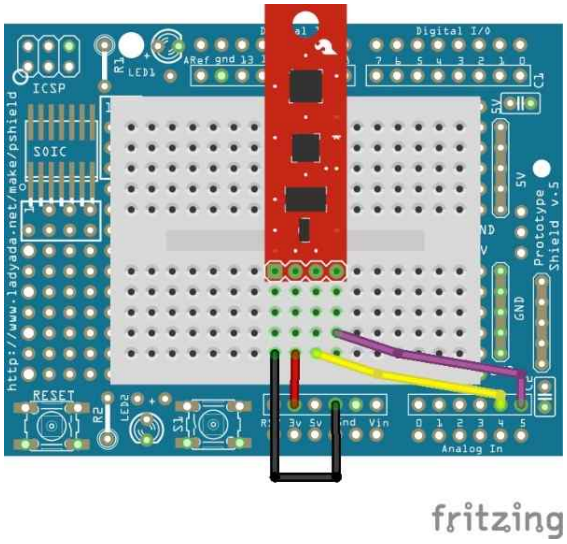


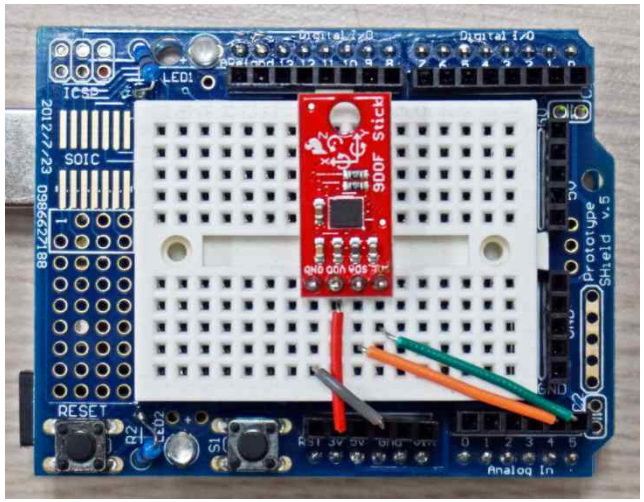
9DoF Sensor connection

- Connect the LSM9DS1 to Arduino.



Pin on LSM9DS1	Pin on Arduino
VCC	3.3V
SCL	A5(SCL)



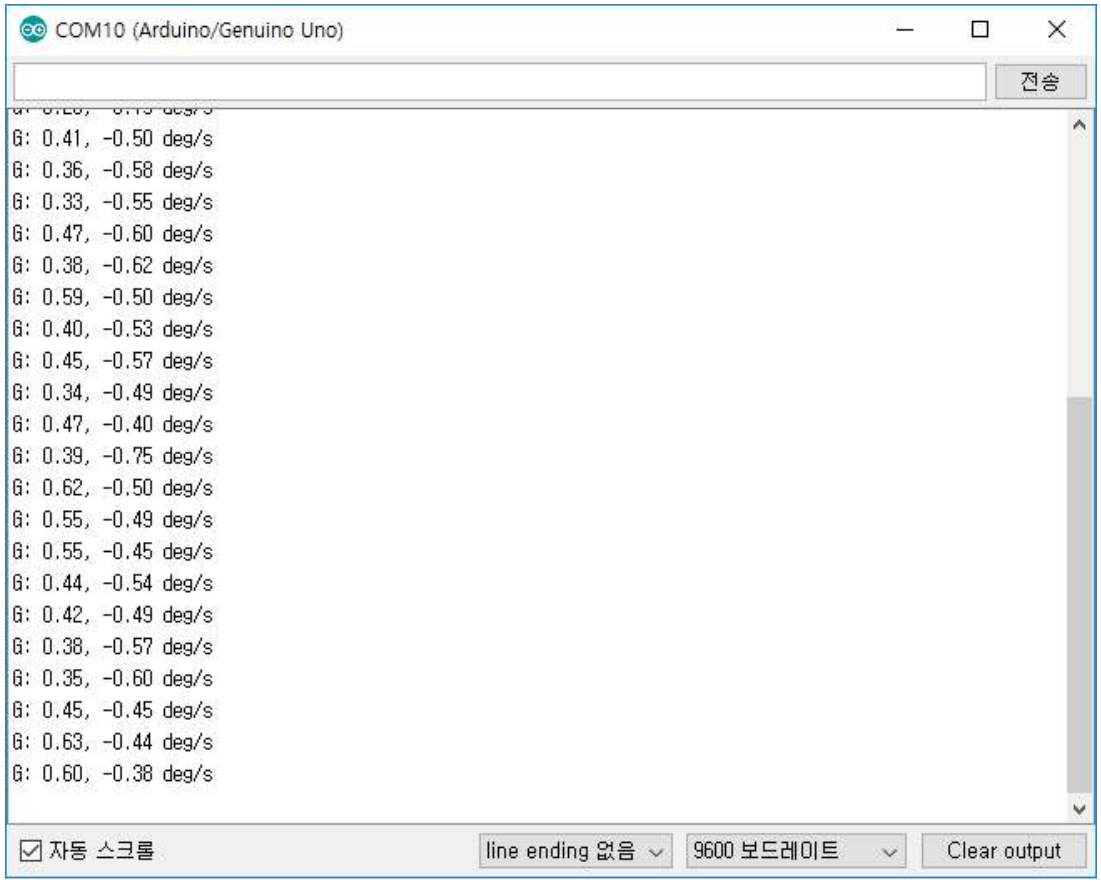


Test of 9DoF IMU sensor

```
#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>

LSM9DS1 imu;

#define LSM9DS1_M 0x1E // Would be 0x1C if SDO_M is LOW
#define LSM9DS1_AG  0x6B // Would be 0x6A if SDO_AG is LOW

void setup() {
  Serial.begin(9600);
  imu.settings.device.commInterface = IMU_MODE_I2C;
  imu.settings.device.mAddress = LSM9DS1_M;
  imu.settings.device.agAddress = LSM9DS1_AG;
  imu.begin();
}
void loop() {
  imu.readGyro();
  Serial.print("G: ");
  Serial.print(imu.calcGyro(imu.gx), 2);
  Serial.print(", ");
  Serial.print(imu.calcGyro(imu.gy), 2);
  Serial.println(" deg/s");
  delay(500);
}
```

Test of 9DoF IMU sensor

- Serial monitor:



A Closer Look

- For I2C, three libraries are used.

```
#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>
```

- imu and parameters are defined for LSM9DS1.

```
LSM9DS1 imu;
#define LSM9DS1_M 0x1E
#define LSM9DS1_AG 0x6B
```

A Closer Look

- Measured values are read by the following command.

```
imu.readGyro();
```

- Gyro sensor measurements are calculated by using the following command.

```
imu.calcGyro(imu.gx)
imu.calcGyro(imu.gy)
```

Angular Displacement Calculation

- Angular rate implies the angular velocity. So, we need an algorithm to calculate the angular displacement from the angular velocity.
- Based on the kinematical relation, $\dot{\theta} = \omega$
- Hence, the angular displacement can be obtained by integrating the angular velocity in time domain.
- In discrete time,

$$\theta_{k+1} = \theta_k + \omega_k \Delta T$$